Hypertext Functionalities with XML

Fabio Vitali
University of Bologna
Department of Computer Science
Mura A. Zamboni, 7
Bologna, Italy
email fabio@cs.unibo.it

ABSTRACT

Hypertext functionalities represent part of the distilled wisdom of the hypermedia community. Given the peculiar nature of the World Wide Web, it is very difficult to successfully propose functionalities that can become widely accepted. Yet,standards such as XLink may provide the needed basic tools to implement most of them. In this paper we briefly discuss the issues brought forth by the hypertext functionalities and introduce XMLC. XMLC is our prototype of an XML browser that, given its modular architecture and general scope, can be used as the basis for implementing sophisticated hypertext functionalities on the Web. Our own implementation of a seminal XLink support is further discussed.

INTRODUCTION

The community of hypertext functionalities [HTF] was born in order to identify and list the functionalities intrinsic to the idea of hypertext, and to either verify them or introduce them in other communities such as the World Wide Web ([VW 98] and [MVW 99]), software engineering [RZ 98], etc.

In particular, the World Wide Web has developed according to ways that were very peculiar and difficult to predict. For instance, the WWW community valued the development of standards and protocols more than functionalities. This has lead to the creation of some dozens of different languages and protocols that are necessary to master the task of creating satisfactory Web sites.

In our opinion this richness of languages shows on the one hand, that there exist the possibility of implementing a large number of interesting functionalities, and on the other hand, that unfortunately the WWW does not enforce or even facilitate them, so that their use depends on the will and awareness of the authors of Web pages and sites.

Furthermore, this richness of possibilities is coming to the detriment of simplicity, which was once the real advantage of the World Wide Web over other systems such as Gopher or FTP. In the near future professional Web authors may have to deal with at least ten different and non-trivial languages or protocols, such as HTML, CSS, ECMAscript (plus any of its proprietary dialects, such as Javascript and Jscript), XML, XSLT, XSL-FO, XPath, XPointer, XLink, XML-Schema, RDF, HTTP, WebDAV, your families of choice of server-side includes, plus many others that at the moment are starting to catch on. So, while there really exist the possibility, in the languages, to provide sophisticated hypertext functionalities, we have to wait for Web applications to actually provide them in a usable way.

Yet, the XML family is a considerable advancement over previous languages and standards. The possibility given by XML [BPS 98] to define a syntax (i.e., a Document Type Definition, or DTD) tailored for one's document classes, and to use standard XML tools to create, verify and exchange data is a real bonus. The strength of XML lies beyond the capabilities to define community-specific DTDs: XML is becoming convenient to use even for application-only data, that is, for objects that are not naturally meant to be displayed to a human user.

Additionally, XSL provides much to XML in terms of reach and flexibility. XSL includes a mapping language [Cla 99] that can be used to transform an XML document into another one. Currently its most important use is to transform an XML document into a format that can be displayed by a browser: thus for instance Microsoft Internet Explorer 5 can accept XML documents of any DTD and use XSL to transform them into an HTML document that can then be properly displayed on a computer screen. Unfortunately this approach is only as flexible as the destination format, i.e. as the language that the final browser understands.

Our long-term purpose is to create an environment that, while relying on most existing Web languages and protocols, can provide fundamental hypertext functionalities in a streamlined and easy way. In past papers we discussed displets ([CRV 98] and [BCV 99]), our proposal to provide flexible support for special rendering needs that authors may have. Displets are software modules (currently they are Java classes) that are associated to each element in an XML document and that provide some behavior (usually rendering behaviors, such as formatting characteristics) for that element. Support for the most common element types is provided (for instance, text elements and paragraphs), but it is possible at any time to add new modules enabling specialized rendering semantics for specific needs.

XMLC can be considered a very general architecture to add sophisticated hypertextual functionalities into documents created in the XML format. The overall design goal is to create a complete authoring environment for sophisticated hypermedia based on the most recent protocols and languages available on the WWW. One of the most interesting of such protocols is XLink [DMO 00], that provides for easy implementation of several sophisticated hypertextual services. In this paper we concentrate on hypothesizing the usefulness of this protocol for the realization of the sophisticated hypertextual functionalities listed in [BVA 97]. Furthermore, we describe how they are being implemented in the current version of our XMLC browser. In fact, the architecture of XMLC can be fruitfully used for more than visualization, for it is an extremely general way to associate behaviors to XML elements, and thus to produce active documents that perform computations, enact goals, produce results.

This paper is structured as follows: in the next section we discuss some of the most important hypertext functionalities on the Web. Next we discuss the current architecture of XMLC, and provide examples of some of the displet classes we have created. Then we discuss how the hypertext functionalities can be implemented using XLink, and how XMLC supports XLink. This makes XMLC a sophisticated architecture for hypertext functionalities using XML.

HYPERTEXT FUNCTIONALITIES ON THE WEB

In [BVA 97] a list of 9 fundamental (according to the authors' opinions) hypermedia functionalities were proposed and discussed, with the understanding that few of them, if any at all, were either available on the World Wide Web or exploited to their full potential:

- Typed nodes and links
- Link attributes and structure-based queries
- Transclusions, warm and hot links
- Annotations and public vs. private links
- Computed personalized links
- External link databases and link update mechanisms
- Global and local overviews
- Trails and guided tours
- Backtracking and history-based navigation.

These items were selected from a longer list of 25 items assembled at the 2nd HTF workshop in conjunction with the Hypertext '96 conference [ABB 96].

At the moment, probably, all of these functionalities could be easily implemented on the WWW. Server-side CGI applications, servlets and DBMSs, as well as client-side plugins, Java and Javascript programs allow now a degree of freedom in customizing the WWW unprecedented in any other hypermedia system (even those that did implement some of these functionalities). The research and commercial communities have in fact already explored some of these functionalities in the last few years. Yet, few of them have really caught on with the larger WWW community, or even found a small visibility stand-point through the available commercial applications.

It is indeed our opinion that no Java applet, CGI application or other custom concoction can possibly produce any relevant change in the way the WWW is used. The reason for this is that these would all be added functionalities to the core sets in servers and browsers, and, unfortunately, the WWW in neither the set of server functionalities, nor the set of browser functionalities.

The fact that the WWW is not a system, or a set of interdependent systems, but a set of protocol and languages, is obvious yet not sufficiently understood. No single system can provide added value to the WWW as a whole. Almost no organization (in many cases not even Microsoft or Netscape) can introduce a new functionality in its products and find out that the WWW as a whole catches on. The WWW must not be improved in the systems, but in the way it actually works: by changing the underlying languages and protocols (HTML, HTTP, CGI, etc.).

We can group the above-mentioned functionalities in two larger families: those that add to the active participation of the users in the production of information, and those that add to the exploration of the available information. On the one hand, annotations, private links and computed personalized links (that require external link bases and link update mechanisms to work on a large scale) allow for the active participation of readers to the nodes they read. On the other hand, overviews, trails, guided tours and sophisticated backtracking patterns (that require richer types and attributes for nodes and links) enhance the navigation and the access to the information of the hyperbase. Finally transclusions and links of various temperature provide both a richer expressive means for authors, and a richer exploration means for readers.

Both families share the same problem: they are not functionalities that can be experienced by the single user, i.e., that one enlightened user can adopt for his/her own purposes and be enriched by using them: they are functionalities that have to be actually used by a large community in order for them to fully provide their benefits: there is little point in using an external link database, if we can't share our links with our colleagues; there is little point in annotating or transcluding, if we can't publish our notes and transclusions; there is little point in being able to create overviews and guided tours on some collection of documents, if we can't publish them for our readers. Thus these functionalities must be dictated through the standards and protocols that make up the Web, rather than through any specific application.

More recently, in [BCV 00] four hypermedia functionalities were further identified:

- editing browsers
- storing document content and link anchors separately
- external linkbases
- displaying link spans, node and link attributes

In all these cases, actual WWW protocols were cited that could provide the necessary expressive power to implement these functionalities: WebDAV [GWF 99] provides clients with remote writing power, thus make editing browsers a real possibility. XPointer [DDM 99] and XLink [DMO 00] allow external links, thus making it possible to separate content and link, and to put links into external linkbases. RDF [BG 00] allows arbitrary meta-information to be added to any Web document, and to be used for classification, indexing, and searches.

A shareable long term goal is to identify a single, simple and streamlined architecture to provide all these functionalities using WWW protocols and hiding the complexities behind the protocols used. With XMLC, which will be described in the next section, we are providing a single, easy to use and easy to expand architecture for browsing XML documents. We consider it a first step in that direction.

XMLC

XSLT [Cla 99] is an important tool to guarantee the generality and flexibility that are the most evident characteristics of XML. XSLT is a mapping language for XML documents, allowing any XML structure to be transformed into another structure by means of contextual rules. Since it is general and rule-based, XSLT allows data creators to determine the ontologies used in their data fragments more or less indipendently of the needs of the data consumers, provided that XSLT rules can be created to transform the fragments.

An important use of XSLT, currently, is to transform XML documents so that they may be shown on an XML browser. Internet Explorer 5, for instance, relies on the rendering model of HTML, and therefore can shown XML documents that can be trasformed into HTML by applying XSL stylesheets. In the near future, XSL-FO (or simply XSL), [ABC 00] will be an alternative rendering model, much more sophisticated than HTML. XSL-FO is being developed as a standard part of the XML family, and provides detailed control of the appearance of text and simple images, providing thus a sophisticated typographical management of XML documents.

The main drawbacks of these approaches is that, on the one hand XML provides sophisticated ways to customize

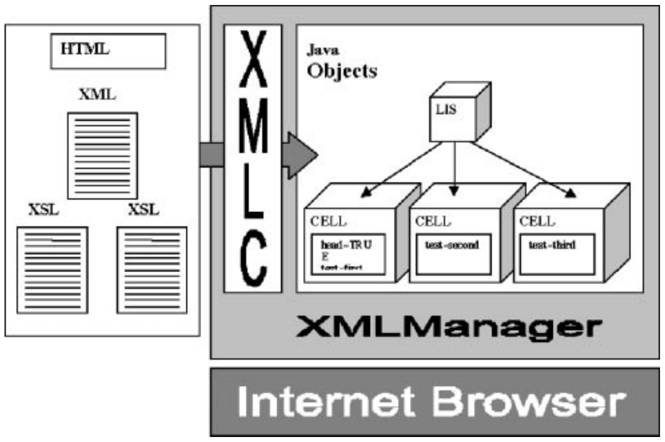


Figure 1 The architecture of the XMLC application.

one's own document models and XSLT provides a very general transformation mechanism for arbitrary XML document, but on the other hand both HTML and XSL-FO will provide a closed set of rendering elements to choose from. Although XSL-FO will be much more sophisticated than HTML, it will still provide only a limited and inextesible set of formatting objects.

Clearly, something is missing. What will be adequate is a generic rendering engine that can select and activate dynamically the set of formatting objects it needs to use, and a way to make this library extensible and open to all sorts of needs. Displets are our proposal for this task. Displets are small independent rendering modules that are used within a modular browser to provide any kind of rendering behavior for XML elements. In our current architecture displets are small and simple JavaBeans that are associated to XML elements by our XMLC application.

The Architecture of XMLC

XMLC (XML Compiler) is our architecture for rendering displets. XMLC relies on technologies and languages such as XML, XSL and DOM, to provide its functionalities.

The main purpose of XMLC is to read an XML document and to produce a displayable tree of Java objects. This happens in a few steps: first, the XML document is read and transformed by a normal XML parser into an internal tree representation based on DOM. Then one or more layers of XSL stylesheets are applied to the DOM tree through the use of an XSLT processor. This creates a final DOM tree that is ready to be displayed. This final DOM tree has an important property: for every element type in the tree there is an available displet to activate. XMLC instantiate all the required displets associating them to the elements of the final DOM tree, and creates thus a tree of runnable objects. Running the root of this tree we obtain an application dynamically generated according to the elements contained in the original document and the associated stylesheets. Figure 1 shows a schema of the architecture.

Each element in the DOM tree is transformed into a displet according to the following rules:

- the DOM element's name determines the Java class to be loaded
- the DOM element's attributes determine the settable properties of the instance of the class
- the DOM element's content (both sub-elements and text content) is added to the tree as children of the class instance.

The current implementation of the XMLC architecture is in Java; a displet can be any sort of Java classes, but using the concept of JavaBeans it is easy to create sophisticated and interoperable displets: the use of JavaBeans Containers and Components, which can be easily organized in hierarchies, nicely fits with the hierarchical nature of DOM trees and XML documents.

Currently, our main use of XMLC is wrapped inside an applet within an HTML document. Parameters of the applet are the XML document to be displayed and the XSL stylesheets to be applied to it. This allows us to display XML documents within well-known Internet browsers. Furthermore, since XML elements are transformed into JavaBeans objects, complex behaviors can be easily added during the lifetime of the visualization, providing support for hypertext jumps, animations, interactions with the reader, and in general all the computational capabilities of the Java language. In the following we briefly report on the simple, display-oriented displets that have been implemented.

Text and Images

We have implemented support for text oriented XML elements. The level of support is comparable to that of HTML 1.0 text elements: basic blocks (P, UL, OL and header elements) and inline chunks (like I, B, TT elements, etc.), plus an image tag and a simple inline hypertextual link. In Table 1 we show a simple HTML document, and show how this is transformed via XSL into a displayable tree.

There are three basic Java displets taking care of the display of text elements: Paragraph, Word and MultiWord. A Paragraph is a container spaced vertically (that is, two or more Paragraphs are put one above the other), with parameterized margins, line height and several other aspects. A Word is a component taking care of the display of a single word (separated by variable-width white space). Words are spaced horizontally and can control font, size, style, baseline and a few other parameters of their content. A MultiWord is a container for Words that is still spaced horizontally. It is used to group together Words that share a common propriety (for instance, that belong to the same run of bold characters, or to the same hypertext anchor).

Managing Layouts

We have developed a displet, called LMXML (Layout Manager for XML), that realizes a flexible layout manager for displets. LMXML was design to provide arbitrary and flexible positioning of text blocks, and to generalize the behaviors of HTML tables, frames and layers.

A few generic containers were created in order to display data regions that:

- can contain any other displets;
- support absolute positioning;
- can be overlapped in transparent or opaque mode;
- their size can be fixed or dynamically computed by contents;
- can be interactively resized by the user;

In Fig. 3 we show regions that partially overlap and have different background transparencies.

The Z Notation

A complete support for the Z notation has been implemented (see also [CVM 99]). The DTD for the notation we use is based on the ZIF Interchange Format [BN 92], although, through the use of different XSL stylesheets, other syntaxes can be used as well.

The support for Z elements is provided through the use of a single displet class, zElement, for all the box types that are present in Z specifications (e.g., schema, axioms, etc.), and a special downloadable font for all the mathematical glyphs specific of the Z language (e.g., function, subset, the set of integers, etc.). All other elements of the Z language are mapped onto plain HTML elements such as P, DIV and SPAN. An additional layer of XSL will then transform them into Paragraph and Word objects as needed.

In Table 3 we show a small fragment of a Z specification (expressed in ZIF) and in Fig. 4 the display of the whole specification in a Web browser.

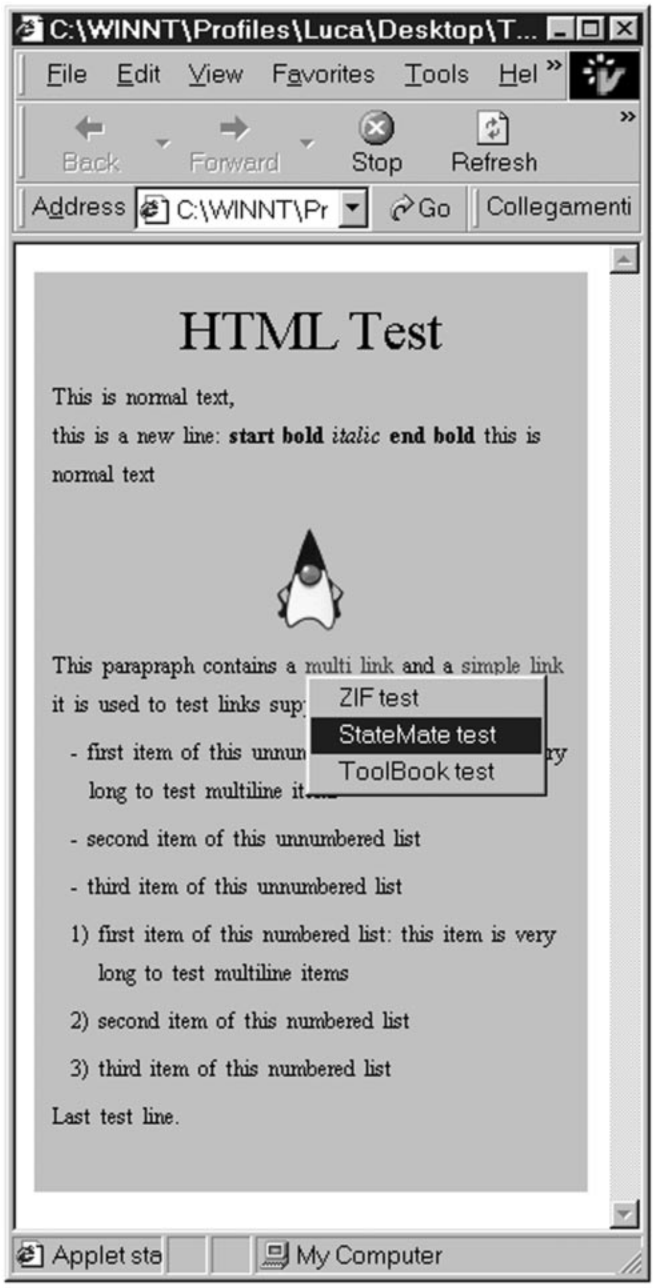
Finite State Machines

A very simple notation for finite state machines has been implemented. Lacking any agreed-upon DTD for it, we have created our own, which is very simple, being composed of just three elements: StateMate (the general container), State (representing a State in the Finite State Machine, shown as a rounded-rect box in the display) and Arc (representing a transition in the Finite State Machine, and shown as an arrow in the display).

Each state has a position and a label, while the arcs have a label but start from and arrive to the center of the state box. The labels are the content of the State and Arc elements, and can be of any kind (that is, one can use any other displet for them, including HTML elements or any other notations, as strange as needed).

Each state has an identifier, which is used by the arcs to identify their origin and destination. States can be initial or final. The author must specify the position of the states, while labels are automatically drawn in the correct position. In Table 4 we provide an example of a simple StateMate fragment, which is then shown in Fig. 5.

Statemate schemas are an example of active displets, since both states and arcs are active. The active state is highlighted, and by clicking either on a transition or on a destination state, it is possible to traverse the available transitions and execute the finite state machine. Non-reachable states and transitions cannot be activated.


C:\WINNT\Profiles\Luca\Desktop\T...
File Edit View Favorites Tools Hel
Back Forward Stop Refresh
Address C:\WINNT\Pr Go Collegamenti
ZIF test
StateMate test
ToolBook test
Applet sta
My Computer

HTML Test

This is normal text,

this is a new line: **start bold *italic* end bold** this is normal text

This parapraph contains a multi link and a simple link it is used to test links sup

- first item of this unnun ry long to test multiline it

- second item of this unnumbered list

- third item of this unnumbered list

1) first item of this numbered list: this item is very long to test multiline items

2) second item of this numbered list

3) third item of this numbered list

Last test line.

Figure 2 An HTML document showing the text displets

Table 1 A fragment of the HTML document in fig. 2

```
<HTML> <BODY>
  ...
  <P>This is normal text, <BR/> this is a new line:
     <B>start bold<I>italic</I> end bold</B> this is
     normal text</P>
  <H1>
    <IMG src="images/java.gif"/>
  </H1>
  ...
</BODY></HTML>
```

Table 2 The same fragment after the XSL transformation

```
<Block>
  ...
  <Paragraph> <Word text="This"/>
    <Word text="is"/>
    <Word text="normal"/>
    <Wordtext="text,"/>
    <NewLine/>
    <Word text="this"/>
    <Wordtext="is"/>
    <Word text="a"/>
    <Word text="new"/>
    <Wordtext="line:"/>
    <Word bold="true" text="start"/>
    <Word bold="true"text="bold"/>
    <Word bold="true" italic="true" text="italic"/>
    <Wordbold="true" text="end"/>
    <Word bold="true" text="bold"/>
    <Wordtext="this"/>
    <Word text="is"/>
    <Word text="normal"/>
    <Wordtext="text"/>
  </Paragraph>
  <Paragraph font-size="30"alignment="CENTER">
    <Picture font-size="30" alignment="CENTER"
             src="images/java.gif"/>
  </Paragraph>
  ...
</Block>
```

TBJava

Toolbook books are closely related to Hypercard stacks, one of the first commercial hypertext systems: books are collections of pages that contain widgets such as buttons, images and text fields. Each object has some default characteristics and can be further enhanced by associating scripts to events that the widget may be subjected to. The scripts are written in a scripting language called OpenScript, that closely resemble scripting language of Hypercard. Toolbook allows very complex applications to be created with the composition of these simple widgets and the specification of fairly simple scripts.

TBJava is a prototype that uses XMLC to display Toolbook books within an Internet browser. TBJava translates Toolbook books in XML documents, and displays and activates them within an XMLC applet. TBJava is composed of four different elements:

- The TBJava DTD, which is the definition of a class of XML documents that match the elements and the properties of a ToolBook book.
- TBK2XML: a ToolBook filter that transforms ToolBook books into TBJava XML documents according to the TBJava DTD
- A Java-based OpenScript interpreter to execute the scripts associated to the objects of the book.
- A set of JavaBeans that implement the ToolBook runtime widget and interact with the OpenScript interpreter.

TBJava relies on XMLC as the runtime environment of the ToolBook book displayed in the Internet browser. XMLC supports all ToolBook objects, their associated properties, behaviors and scripts. Furthermore, it proposes a set of language-independent and modular paradigms for message passing and event handling. This allows ToolBook books to include transparently both ToolScript and Javascript scripts.

HYPERTEXT FUNCTIONALITIES WITH XMLC

Hypertext functionalities revolve around the sophistication of the concept of link, which is particularly limited in the form provided with HTML. All the elements of the list shown in section 2 constitute a sophistication of the linking model presented in HTML.

W3C is proposing two languages to express hypertext links in XML. XPointer [DDM 99] provides a way to express sub-resource addresses within XML documents and other resources, and XLink [DMO 00] defines a syntax for hypertextual links between XML documents.

XPointers can specify locations within XML documents by collecting progressively detailed location specifiers. This makes it possible to specify an arbitrarily small location without marking it with a tag as in HTML. XPointer makes use of XPath addresses to refer to subtrees of the XML document, and extends it providing more specific address types, such as points and ranges.

XLinks extends HTML links by introducing several new features:

- Links can refer to multiple end-points;
- Links can be multi-directional;
- Links can be stored externally to the resources they link;
- Links can be activated in a variety of ways (they may

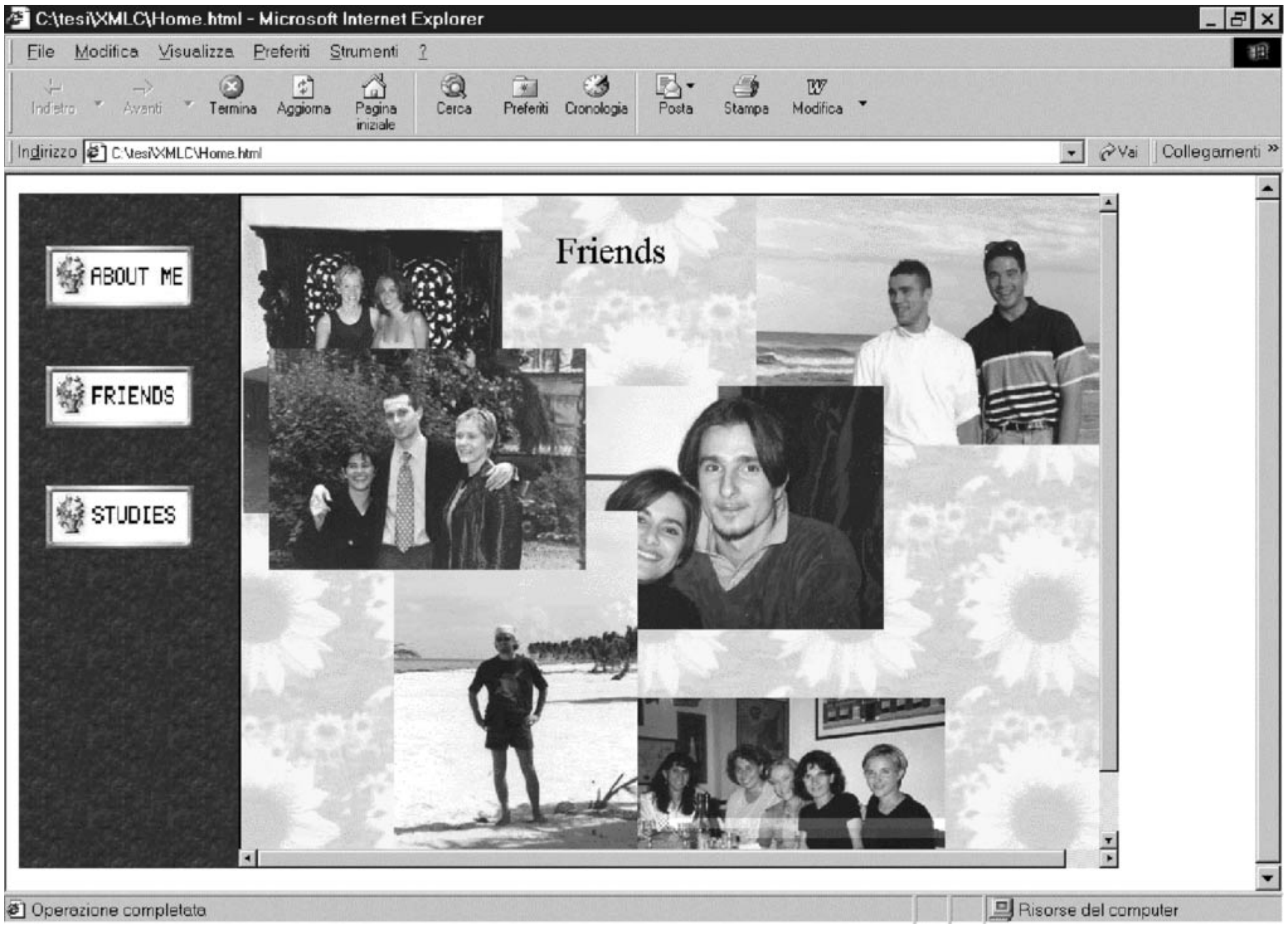

C:\tesi\XMLC\Home.html - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro Avanti Termina Aggiorna Pagina iniziale Cerca Preferiti Cronologia Posta Stampa Modifica
Indirizzo C:\tesi\XMLC\Home.html
Vai Collegamenti
ABOUT ME
FRIENDS
STUDIES
Friends
Operazione completata
Risorse del computer

Figure 3 A complex layout with LMXML

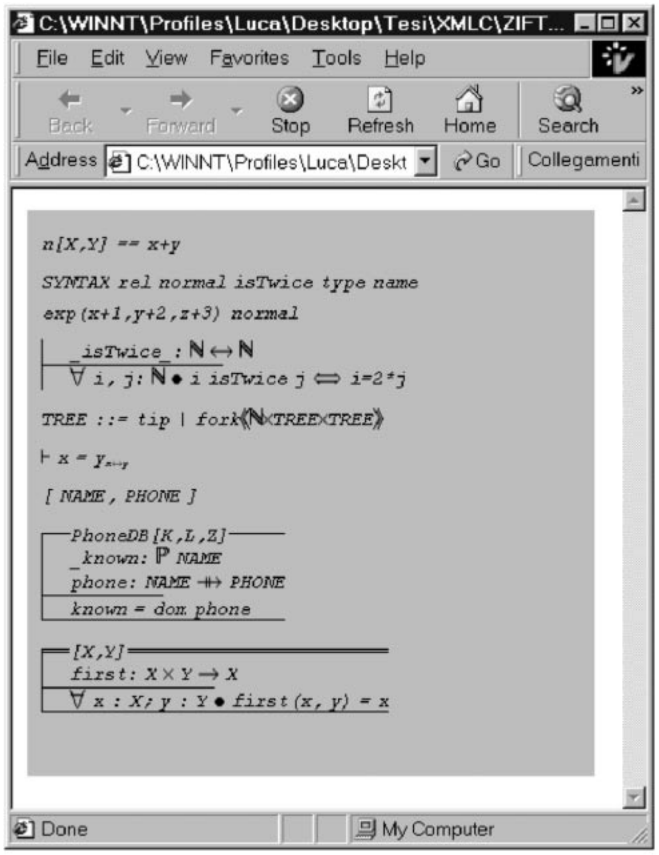
open a new window, substitute the current content, or expand within the current content, etc)
- Links can create groups of related documents that can for instance, be loaded together.

Most of the functionalities present in the list at the beginning of section 2 can be most probably created with the aid of XPointer and XLink. For instance:

- *Typed nodes and links*: since XML and XLink do not impose names on content elements and link elements, it is very easy to create typed nodes and links, allowing document authors to supply the names that would type these entities.
- *Transclusions*: XLink provides a way to specify behavioral attributes that allow links, among other properties, to embed their destinations in place of their sources, and furthermore to activate the embedding at loading time. This easily allows the creation of virtual documents that include fragments of other documents through transclusion, so that the content is always updated to the latest version of the transcluded documents. XLink, on the other hand, does not provide any mechanism for verifying the correctness of the transclusion, which would lead to what in literature is known as the problem

Table 3 A small fragment of a Z specification

```
...<schemadef style="vert" purpose="state">
  PhoneDB
<decpart>
    <declaration> _known: &pset; NAME </declaration>
    <declaration> phone: NAME &fpfun; PHONE </declaration>
  </decpart>
  <formals> K,L,Z </formals>
<axpart>
    <predicate> known = &dom; phone </predicate>
</axpart>
</schemadef>
...
```


C:\WINNT\Profiles\Luca\Desktop\Tesi\XMLC\ZIFT...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address C:\WINNT\Profiles\Luca\Deskt Go Collegamenti
Done My Computer

$n[X,Y] == x+y$

SYNTAX rel normal isTwice type name

exp(x+1,y+2,z+3) normal

$\_isTwice\_ : \mathbb{N} \leftrightarrow \mathbb{N}$

$\forall\ i, j : \mathbb{N} \bullet i\ isTwice\ j \Leftrightarrow i=2*j$

$TREE ::= tip \mid fork\langle\langle \mathbb{N} \times TREE \times TREE \rangle\rangle$

$\vdash x = y_{x \mapsto y}$

$[\ NAME, PHONE\ ]$

$PhoneDB[K,L,Z]$

$\_known : \mathbb{P}\ NAME$

$phone : NAME \mathrel{+\!\!\!\twoheadrightarrow} PHONE$

$known = \mathrm{dom}\ phone$

$[X,Y]$

$first : X \times Y \rightarrow X$

$\forall\ x : X; y : Y \bullet first(x, y) = x$

Figure 4 The visualization of the Z specification

Table 4 A simple finite state machine in XML format

```
<StateMate>
  LEVEL_MANAGER_CONTROL
  <Arc from="state2" to="state1">
    LM_ACTIVE
  </Arc>
  <Arc from="state1" to="state3">
    LM_MORE
  </Arc>
  <Arc from="state3" to="state2">
    UPDATE
  </Arc>
  <State id="state1" start="true" origin="0,50">
    WAIT
  </State>
  <State id="state2" origin="250,50">
    NEW_VAL
  </State>
  <State id="state3" end="true" origin="100,200">
    PUMP_ACTIVE
  </State>
</StateMate>
```

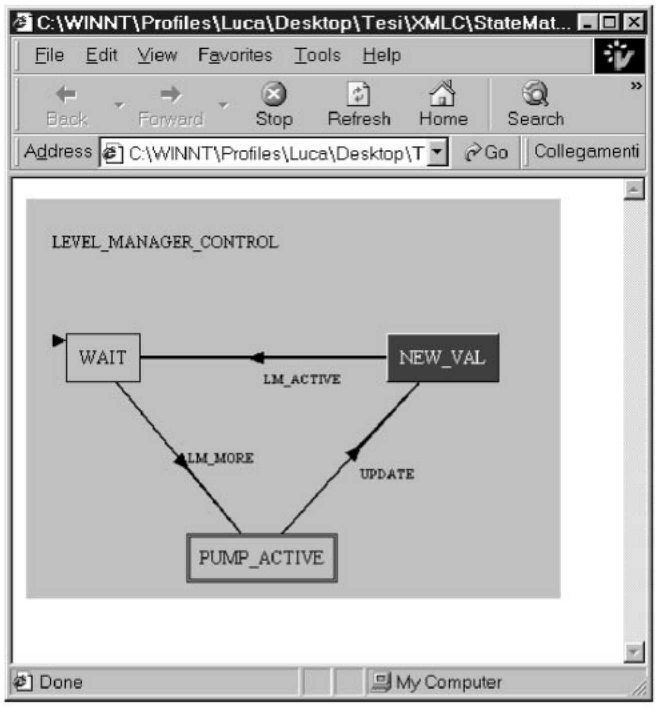

C:\WINNT\Profiles\Luca\Desktop\Tesi\XMLC\StateMat...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address C:\WINNT\Profiles\Luca\Desktop\T Go Collegamenti
LEVEL_MANAGER_CONTROL
WAIT
NEW_VAL
LM_ACTIVE
LM_MORE
UPDATE
PUMP_ACTIVE
Done My Computer

Figure 5 The representation of the finite state machine of table 3

referential integrity of links [Dav 98]. In [Vit 00] a justification for using versioning to solve this problem is shown.

External linkbases: XLink explicitly mentions linkbases as possible repositories for external extended links, and calls for an out-of-band method for specifying their locations. Nonetheless, it defines the "external-linkset" role in order to help document authors to specify the location of possible link repositories that apply to their documents. XLink, thus, explicitly supports external linkbases.

Overviews, trails and guided tours: as mentioned in [GSB 00], XLink does not explicitly support the creation of ordered collections of links that could shape and organize the navigation through a complex maze of documents by providing overviewsm trails and guided tours. Nonetheless, an appropriate use of external linkset could easily allow restricted link sets to be created and used as overviews , trails and guided tours.

In [GSB 00] one other limitation of XPointer was pointed out, and namely the impossibility of XPointer to refer to location in non-XML documents. The Open Hypermedia community [OHSWG] has often discussed the issue of actually integrating hypermedia with our daily tools, and referencing non-XML content is an importat issue in this community. Their proposals (e.g. [Gro 98]) could easily be integrated in the syntax of XPointers, so that a standard

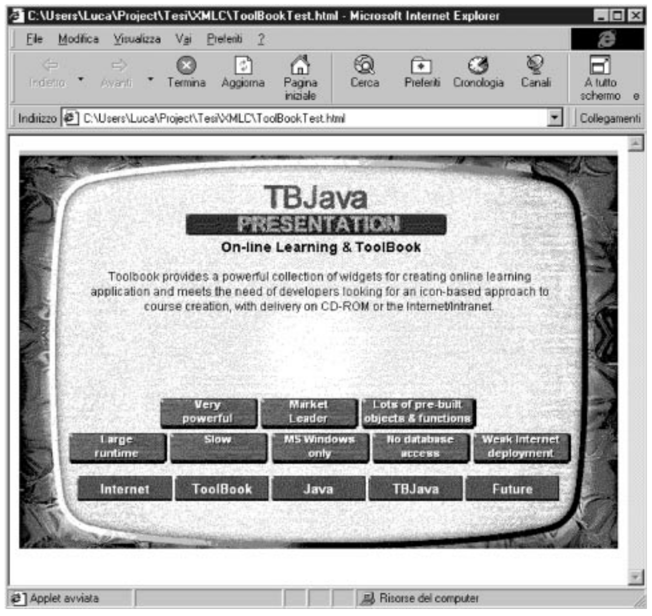
Table 5 a Toolbook book as an XML document

```
<?xml version="1.0"?>
<Book size="9000,6000">
  <Background size="9000,6000">
    <Page name="page1">
      <Button bounds="4500,4100,6000,4600" caption="page2">
        <Script type="OpenScript">
          <Handler type="to handle buttonClick">
            to handle buttonClick
              go to page 2
            end
          </Handler>
        </Script>
      </Button>
      <Field bounds="1000,1500,8000,1200" text="page 1"
             textAlignment="center"/>
    </Page>
    <Page name="page2">
      <Button bounds="4500,4100,6000,4600" caption="page1">
        <Script type="OpenScript">
          <Handler type="to handle buttonClick">
            to handle buttonClick
              go to page 1
            end
          </Handler>
        </Script>
      </Button>
     <Field bounds="1000,1500,8000,1200" text="page 2"
            textAlignment="center"/>
    </Page>
  </Background>
</Book>
```

referencing mechanism could be provided for all types of media.

Our research group has provided a basic implementation of XLink based on displets for our XMLC architecture. This has added a few steps to the sequence of transformations of the XMLC application, as shown in Fig. 7. The XMLC applications does not simply add the document to be displayed and the XSL stylesheet, but it also looks for additional XML documents forming a document group. These documents could contain XLinks that need to be added to the document before displaying it.

After parsing the XML documents, all link elements are identified and added to a list. Then, an identifier is added to all the addressable elements of the document, since after the application of the XSL stylesheets the structure of the document can become arbitrarily different from the original one, and it is necessary to provide a way to identify the elements that can be located through XPointers. The document then are subjected to the usual XSL transformations. Before displaying, though, additional wrapper classes are added around


C:\Users\Luca\Project\Tesi\XMLC\ToolBookTest.html - Microsoft Internet Explorer
File Modifica Visualizza Vai Preferiti ?
Indietro Avanti Termina Aggiorna Pagina iniziale Cerca Preferiti Cronologia Canali A tutto schermo
Indirizzo C:\Users\Luca\Project\Tesi\XMLC\ToolBookTest.html
Collegamenti
TBJava
PRESENTATION
On-line Learning & ToolBook
Very powerful
Market Leader
Lots of pre-built objects & functions
Large runtime
Slow
MS Windows only
No database access
Weak Internet deployment
Internet
ToolBook
Java
TBJava
Future
Applet avviata
Risorse del computer

Toolbook provides a powerful collection of widgets for creating online learning application and meets the need of developers looking for an icon-based approach to course creation, with delivery on CD-ROM or the Internet/Intranet.

Figure 6 The representation of a Toolbook book in a Web browser

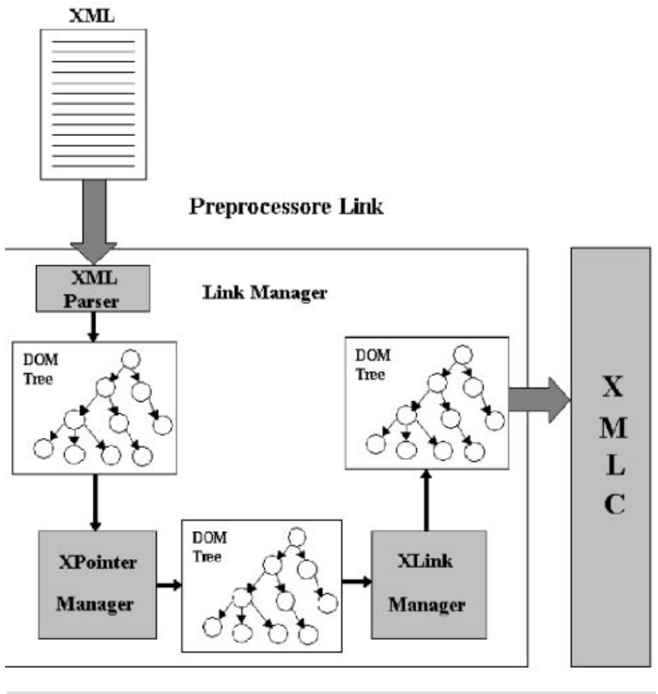


Figure 7 The XLink-enabled architecture of XMLC

the document elements that are starting points of links, to provide the most appropriate navigational functionality. When the user clicks on one such element, the class reacts, consults the list of destinations, and activates the navigation.

The implemented management of document groups is quite sophisticated and implements the 'show' attribute of

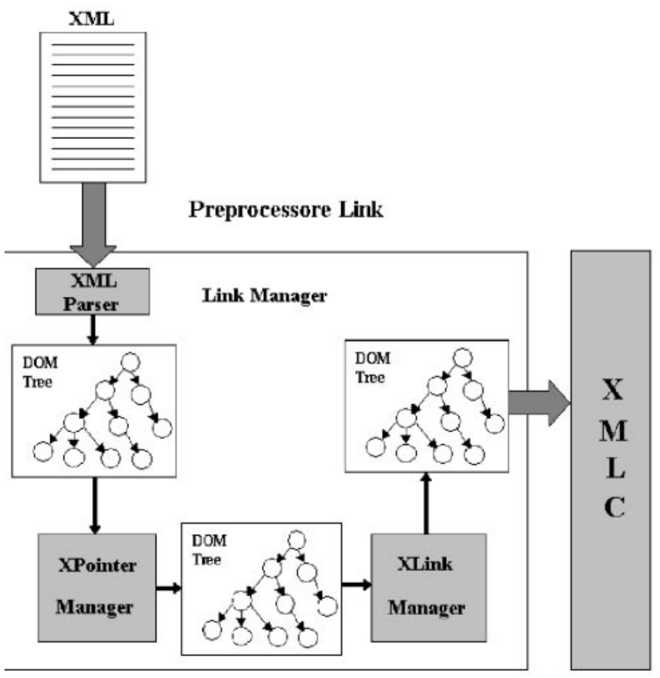


Figure 8 A simple hyperlinked document group

XLink, controlling whether the destination document will replace the current one, or it will be created in a new window, or it will integrate with the current document. Fig. 8 shows a sample hyperlinked document group.

CONCLUSIONS

Hypertext functionalities will be slow in implementation, and even slower in acceptance. It is just too difficult to take care of them by non-professionals. Furthermore the Web, born as an exchange medium for professionals, has clearly become a graphic, impact-oriented one-way medium for the presentation of corporate truths or inflated egos.

The kind of ideas and functionalities presented here and in the literature on hypermedia functionalities present important characteristics anyway, that we presume will become more and more important as the public gets acquainted with the possibilities of the new medium.

Yet, in order to provide easily sophisticated functionalities as the ones mentioned, the current architecture of the clients and the servers needs to be rethought. In particular, fewer and more powerful protocols and standards need to be used.

The XML family is in our opinion an important step in that direction. XML and its cohort can actually let users and authors express their data and wishes in a sophisticated, customizable and expandable way. But a new software architecture needs to be implemented to take advantage of the generality of these languages.

XMLC in our opinion is the most customizable and expandable architecture for displaying XML documents. Being expandable, it has been easy to add support for several sophisticated hypertext functionalities, such as the ones allowed by XLinks and XPointers. Word is under way to add more of them to future implementations.

XMLC is a working prototype, and can be examined, downloaded and used. We gladly point the interested reader to the URL:*http://www.cs.unibo.it/projects/displets/*

We would like to acknowledge here the contribution of all the people that have worked on this architecture: Michael Bieber, Paolo Ciancarini, Chao-Min Chiu, Cecilia Mascolo, Stefano Pancaldi, Alfredo Rizzi, Alessandro Rocca, Alessandro Ronchi, Silvia Villa, and all the students of the undergraduate course in Software Engineering at the Computer Science Department of the University of Bologna. We also wish to thank Microsoft for its partial support to this research topic.

BIBLIOGRAPHY

[**ABB 96**] H. Ashman, V. Balasubramanian, M. Bieber, H. Oinas-Kukkonen (Eds.), Proceedings of the 2nd International Workshop on Incorporating Hypertext Functionality into Software Systems (HTFII), Hypertext '96 Conference, Washington, 1996, http://www.cs.nott.ac.uk/~hla/HTF/HTFII/Proceedings.html

[**ABC 00**] S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles, Extensible Stylesheet Language (XSL) 1.0, W3C Working Draft 27 March 2000, http://www.w3.org/TR/xsl

[**BCV 99**] L. Bompani, P. Ciancarini, F. Vitali, "Active Documents in XML", ACM SigWeb Newsletter 8 (1), 1999, p. 27–32.

[**BCV 00**] L. Bompani, P. Ciancarini, F. Vitali, Software Engineering and the Internet: a roadmap, in "The Future of Software Engineering", A. Finkelstein (ed.), ACM Press, in print.

[**BPS 98**] T. Bray, J. Paoli, C. M. Sperberg-McQueen, Extensible Markup Language, (XML) 1.0, W3C Recommendation 10 February 1998, http://www.w3.org/TR/REC-xml

[**BVA 97**] M. Bieber, F. Vitali, H. Ashman, V. Balasubramanian, H. Oinas-Kukkonen, "Fourth Generation Hypertext: Some Missing Links for the World Wide Web", International Journal of Human-Computer Studies 47, Academic Press, 1997, p. 31–65.

[**BG 00**] D. Brickley, R.V. Guha, Resource Description Framework (RDF) Schema Specification, W3C Candidate Recommendation, 27 March 2000, http://www.w3.org/TR/rdf-schema

[**BN 92**] S. Brien, J. Nicholls, Z Base standard, Programming Research Group, Oxford, UK, 1992

[**Cla 99**] J. Clark, XSL Transformation (XSLT) 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/xslt

[**CRV 98**] P. Ciancarini, A. Rizzi and F. Vitali, "An Extensible Rendering Engine for XML and HTML", Computer Networks and ISDN Systems, 30(1–7):225–238, 1998.

[**CVM 99**] P. Ciancarini, F. Vitali, C. Mascolo, "Managing complex documents over the WWW: a case study for XML", IEEE Transactions on Knowledge and Data Engineering 11 (4), 1999, p. 629–638.

[**Dav 98**] H. C. Davis, Referential Integrity of Links in Open

Hypermedia Systems, in Hypertext 98 Proceedings (Pittsburgh, PA), ACM Press, New York, 207-216

[**DDM 99**] S. DeRose, R. Daniel Jr., XML Pointer Language (XPointer), W3C Working Draft, 6 December 1999, http://www.w3.org/TR/xptr

[**DMO 00**] S. DeRose, E. Maler, D. Orchard, B. Trafford, XML Linking Language (XLink), World Wide Web Consortium Working Draft 21 February 2000, http://www.w3.org/TR/xlink

[**GWF 99**] Y. Goland, E. Whitehead, A. Faizi, S. Carter, D. Jensen, HTTP Extensions for Distributed Authoring – WEBDAV, IETF RFC 2518, February 1999, http://www.ietf.org/rfc/rfc2518.txt

[**Gro 98**] K. Grønbaek, OHS interoperability—issues beyond the protocol, Proceedings of OHS Workshop 4.0, Hypertext 98, Pittsburgh, 1998

[**GSB 00**] K. Grønbaek, L. Sloth, N.O. Bouvin, Open Hypermedia as User Controlled Meta Data for the Web, in 9th Internetional World Wide Web Conference Proceedings, Amsterdam, Holland, 2000, p. 553–566

[**HTF**] The Hypertext Functionality Workshop series, http://www.cs.nott.ac.uk/~hla/HTF/

[**MVW 99**] M. Milosavljevic, F. Vitali, C. Watters, Proceedings of the Workshop on Virtual Documents, Hypertext Functionalities and the Web, VIII Int. World Wide Web Conference, Toronto, Canada, 1999, http://www.cs.unibo.it/~fabio/VD99/

[**OHSWG**] The Open Hypermedia Systems Working Group, http:/ /www.ohswg.org/

[**RZ 98**] G. Rossi, H. Ziv (eds.), Proceedings of the Fifth International Workshop on Engineering Hypertext Functionality into Future Information Systems (HTF5), ICSE 98 Conference, Kyoto, 1998, http://www.ics.uci.edu/pub/kanderso/htf5/papers/

[**VB 00**] F. Vitali, M. Bieber, “Hypermedia on the Web: What Will It Take?”, ACM Computing Survey, in print.

[**VW 98**] F. Vitali, C. Watters (eds.), Proceedings of the Fourth International Workshop on Hypertext functionality and the WWW (HTF4), 7th Int. WWW Conference, Brisbane, 1998, http://dragon.acadiau.ca/~cwatters/htf4/

[**Vit 00**] F. Vitali, Versioning Hypermedia, ACM Survey, 2000, in print.

BIOGRAPHY

Fabio Vitali is assistant professor at the Department of Computer Science of the University of Bologna. He holds a Laurea degree in Mathematics and a PhD in Computer and Law, both from the University of Bologna. His research interests include markup languages, distributed, coordinated systems, and the World Wide Web. He is author of several papers on hypertex functionalities, the World Wide Web and XML.